RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY

Sommersemester 2003
29.04.2003 (Abgabe: 06.05.2003)

Prof. Dr. K. Indermark
Dr. Th. Noll

1. Übung *Programmanalyse und Compileroptimierung*

Erinnerung: Am 30. April findet keine Frontalübung statt. Das vorliegende Aufgabenblatt wird am 7. Mai vorgerechnet.

Aufgabe 1

Wenden Sie die Optimierung der Dead Code Elimination auf das folgende SLC–Programm an. Geben Sie hierzu auch die LV–Mengen aus der Live Variables–Analyse an:

$$\begin{aligned}\pi = (\beta, IV, OV) \in SLC \quad \text{mit} \quad IV &: y, z;\\ \beta &: x \leftarrow y + z;\\ &\quad\ y \leftarrow y + z;\\ &\quad\ x \leftarrow y - z;\\ &\quad\ y \leftarrow x + z;\\ &\quad\ z \leftarrow x * y;\\ OV &: x, y\end{aligned}$$

Aufgabe 2

Weisen Sie nach, dass die Dead Code Elimination eine idempotente Transformation des betreffenden SLC–Programms darstellt, d.h. dass nach einmaliger Anwendung von T_{dc} keine überflüssigen Anweisungen mehr existieren:

$$T_{dc}(T_{dc}(\pi)) = T_{dc}(\pi)$$

für alle $\pi \in SLC$.

Aufgabe 3

Wenden Sie die Optimierung der Common Subexpression Elimination auf das folgende SLC–Programm an. Geben Sie hierzu auch die AE– und die wh–Mengen aus der Available Expressions–Analyse an:

$$\begin{aligned}\pi = (\beta, IV, OV) \in SLC \quad \text{mit} \quad IV &: u, y, z;\\ \beta &: x \leftarrow y * z;\\ &\quad\ u \leftarrow u - x;\\ &\quad\ v \leftarrow y * z;\\ &\quad\ z \leftarrow u + v;\\ &\quad\ w \leftarrow y * z;\\ &\quad\ u \leftarrow u + v;\\ &\quad\ v \leftarrow y * z;\\ OV &: u, v, w.\end{aligned}$$

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY

Sommersemester 2003
06.05.2003 (Abgabe: **09.05.2003**)

Prof. Dr. K. Indermark
Dr. Th. Noll

2. Übung *Programmanalyse und Compileroptimierung*

Hinweise:

- Da der in der Vorlesung präsentierte Stoff zur vollständigen Bearbeitung von Ü1A3 (Common Subexpression Elimination) nicht ausreichte, wird diese bereits dann gewertet, wenn nur die AE–Mengen berechnet wurden.
- Um den zeitlichen Abstand zwischen der Bearbeitung und dem Vorrechnen der Aufgaben zu verringern, werden ab sofort die Aufgabenblätter und -lösungen freitags ausgegeben bzw. eingesammelt. Dies gilt bereits für das vorliegende (verkürzte) 2. Aufgabenblatt: Abgabe ist kommenden Freitag (9. Mai), Vorrechnen übernächsten Mittwoch (14. Mai).

Aufgabe 4

Wenden Sie die Optimierung Konstantenfaltung auf das folgende SLC–Programm an. Geben Sie hierzu auch die RD–Mengen aus der Reaching Definitions–Analyse an:

$$\pi = (\beta, IV, OV) \in SLC \quad \text{mit} \quad \begin{aligned} IV &: z; \\ \beta &: x \leftarrow 3; \\ & \quad y \leftarrow x - 1; \\ & \quad x \leftarrow x * z; \\ & \quad z \leftarrow x - 1; \\ & \quad w \leftarrow y * 3; \\ OV &: w \end{aligned}$$

Aufgabe 5

Bei der *Entfernung von Kopien* handelt es sich um eine optimierende Programmtransformation, die z.B. folgendes Ergebnis liefert:

$$\begin{array}{l} x \leftarrow y; \\ z \leftarrow x + 1 \end{array} \quad \Longrightarrow \quad \begin{array}{l} x \leftarrow y; \\ z \leftarrow y + 1 \end{array}$$

Dahinter verbirgt sich die Idee, nach der Kopieranweisung $x \leftarrow y$ nach Möglichkeit y statt x zu benutzen. Hierdurch wird die Kopieranweisung günstigstenfalls zu „Dead Code“ und kann anschließend eliminiert werden.
Geben Sie eine Analyse zur Erkennung von Kopien sowie eine darauf aufbauende Programmtransformation an.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY

Sommersemester 2003
09.05.2003 (Abgabe: 16.05.2003)

Prof. Dr. K. Indermark
Dr. Th. Noll

3. Übung *Programmanalyse und Compileroptimierung*

Aufgabe 6

Bestimmen Sie die DAG–optimierte Version des folgenden SLC–Programms und vergleichen Sie das Resultat mit demjenigen Programm, das sich nach Anwendung von Konstantenfaltung (T_{cf}), Dead Code–Elimination (T_{dc}) und Common Subexpression–Elimination (T_{cs}; in dieser Reihenfolge) ergibt:

$$\begin{aligned}\pi = (\beta, IV, OV) \in SLC \quad \text{mit} \quad IV &: x, y;\\ \beta &: u \leftarrow 3;\\ &\quad v \leftarrow x - y;\\ &\quad w \leftarrow u + 1;\\ &\quad x \leftarrow x - y;\\ &\quad u \leftarrow x - y;\\ &\quad z \leftarrow u * w;\\ &\quad u \leftarrow 2 * u;\\ &\quad v \leftarrow w - 1;\\ OV &: u, v\end{aligned}$$

Aufgabe 7

Zeigen Sie, dass ein DAG–optimiertes Programm bzgl. Konstantenfaltung, Dead Code– und Common Subexpression–Elimination optimal ist, d.h. dass es durch die Transformationen T_{cf}, T_{dc} und T_{cs} nicht mehr verändert wird.

Aufgabe 8

In der Vorlesung wurde die Frage aufgeworfen, ob die DAG–Optimierung durch eine (längenbeschränkte) Folge optimierender Einzeltransformationen dargestellt werden kann. In dieser Aufgabe soll gezeigt werden, dass die einmalige Anwendung der Common Subexpression–Elimination (und der Entfernung von Kopien; vgl. Ü2A5) nicht ausreicht, um die DAG–optimierte Version eines SLC–Programms zu erhalten. Wir betrachten hierzu das folgende Programm:

$$\begin{aligned}\pi = (\beta, IV, OV) \in SLC \quad \text{mit} \quad IV &: x, y, z;\\ \beta &: t \leftarrow x + y;\\ &\quad v \leftarrow z + t;\\ &\quad u \leftarrow x + y;\\ &\quad w \leftarrow z + u;\\ OV &: v, w\end{aligned}$$

(a) Bestimmen Sie die DAG–optimierte Version π_D von π.

(b) Zeigen Sie, dass die schrittweise Transformation von π in π_D die zweimalige Anwendung der Common Subexpression–Elimination (inkl. Entfernung von Kopien) erfordert.

(c) Konstruieren Sie eine Folge $(\pi_n)_{n\in\mathbb{N}}$ von SLC–Programmen, deren DAG–Optimierung jeweils n Anwendungen der Common

Subexpression–Elimination umfasst.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY

Sommersemester 2003
16.05.2003 (Abgabe: 23.05.2003)

Prof. Dr. K. Indermark
Dr. Th. Noll

4. Übung *Programmanalyse und Compileroptimierung*

Aufgabe 9

Seien $\langle D; \leq \rangle$ eine Halbordnung und $T, U \subseteq D$ so, dass sowohl $\bigsqcap U$ als auch $\bigsqcap(T \cup U)$ existieren. Zeigen Sie, dass gilt:

$$\bigsqcap(T \cup U) = \bigsqcap\left(T \cup \left\{\bigsqcap U\right\}\right)$$

Aufgabe 10

Beweisen Sie, dass für jede Halbordnung $\langle D; \leq \rangle$ die Aussagen

(a) $\langle D; \leq \rangle$ ist ein vollständiger Verband,

(b) jede Teilmenge von D besitzt eine kleinste obere Schranke und

(c) jede Teilmenge von D besitzt eine größte untere Schranke

äquivalent sind.

Aufgabe 11

Beweisen Sie, dass stetige Funktionen unter Komposition abgeschlossen sind, d.h. dass für alle vollständigen Halbordnungen $\langle D; \leq \rangle$ und alle stetigen Funktionen $f, g : D \to D$ gilt, dass auch $g \circ f : D \to D : d \mapsto g(f(d))$ stetig ist.

Aufgabe 12

Zeigen Sie, dass für alle $n \in \mathbb{N}$ die Funktion $f_n : \mathfrak{P}(\mathbb{N}) \to \mathfrak{P}(\mathbb{N})$ mit

$$f_n(T) := \begin{cases} T & \text{falls } T \text{ endlich} \\ T \cup \{n\} & \text{sonst} \end{cases}$$

auf der vollständigen Halbordnung $\langle \mathfrak{P}(\mathbb{N}); \subseteq \rangle$ monoton, aber nicht stetig ist.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY

Sommersemester 2003
23.05.2003 (Abgabe: 30.05.2003)

Prof. Dr. K. Indermark
Dr. Th. Noll

5. Übung *Programmanalyse und Compileroptimierung*

Aufgabe 13

Ein weiteres, bisher nicht diskutiertes Kostenmaß ist der *Speicherplatzbedarf* zur Laufzeit eines Programms. Dieser lässt sich für SLC–Programme $\pi \in SLC$ etwa durch die Kostenfunktion $c(\pi) := |V_\pi|$ abschätzen.

Entwickeln Sie eine Optimierung (d.h. eine geeignete Analyse sowie eine darauf aufbauende Programmtransformation), welche den Speicherplatzbedarf eines SLC–Programms reduziert. Diese könnte z.B. das Programm

$$\begin{array}{l} IV : x; \\ \quad \beta : u \leftarrow x + 1; \\ \qquad v \leftarrow 2 * u; \\ \qquad y \leftarrow v; \\ OV : y \end{array} \qquad \text{transformieren in} \qquad \begin{array}{l} IV : x; \\ \quad \beta : x \leftarrow x + 1; \\ \qquad x \leftarrow 2 * x; \\ \qquad y \leftarrow x; \\ OV : y \end{array}$$

und dadurch die Variablen u und v einsparen.

Aufgabe 14

Zeigen Sie anhand des Fakultätsprogramms der Vorlesung, dass das Einsetzungsfunktional des Gleichungssystems zur Fortsetzungssemantik mehrere Fixpunkte besitzen kann.

(Zur Erinnerung: Das (vereinfachte) Gleichungssystem besitzt die Gestalt

$$\begin{aligned} F(x) &= G(x, 1, 0) \\ G(x, y, z) &= \textsf{if } x = z \textsf{ then } y \textsf{ else } G(x, y * (z + 1), z + 1) \end{aligned}$$

und ist in dem Funktionenraum

$$FR := (\mathbb{Z} \dashrightarrow \mathbb{Z}) \times (\mathbb{Z}^3 \dashrightarrow \mathbb{Z})$$

zu lösen.)

Aufgabe 15

(a) Geben Sie ein IC–Programm $\pi \in IC$ zur Berechnung der Fibonaccifunktion an.

(Zur Erinnerung: Die Fibonaccifunktion

$$f : \mathbb{N} \to \mathbb{N}$$

ist rekursiv definiert durch

$$\begin{aligned} f(0) &:= 0, \\ f(1) &:= 1 \text{ und} \\ f(n) &:= f(n-1) + f(n-2) \text{ für alle } n \geq 2.) \end{aligned}$$

(b) Konstruieren Sie das Flussdiagramm sowie den Flussgraphen zu π.

(c) Bestimmen Sie die Fixpunktsemantik von π.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY

Sommersemester 2003
30.05.2003 (Abgabe: 06.06.2003)

Prof. Dr. K. Indermark
Dr. Th. Noll

6. Übung *Programmanalyse und Compileroptimierung*

Aufgabe 16

Wenden Sie die Datenflussanalyse zur Konstantenfaltung, wie in der Vorlesung beschrieben, auf das folgende IC–Programm an:

$$
\begin{array}{rl}
 & \mathsf{in}\ x; \mathsf{out}\ w; \mathsf{loc}\ y, z; \\
1: & y \leftarrow 0; \\
2: & z \leftarrow 1; \\
3: & \mathsf{if}\ x = 0\ \mathsf{goto}\ 11; \\
4: & x \leftarrow x + 1; \\
5: & \mathsf{if}\ x = 1\ \mathsf{goto}\ 9 \\
6: & y \leftarrow 1; \\
7: & z \leftarrow 1; \\
8: & \mathsf{goto}\ 3; \\
9: & y \leftarrow 2; \\
10: & \mathsf{goto}\ 3; \\
11: & w \leftarrow y * z
\end{array}
$$

(a) Konstruieren Sie den entsprechenden Flussgraphen.

(b) Stellen Sie die Gleichungen zur Bestimmung der *RD*–Mengen nach der MFP–Methode auf.

(c) Bestimmen Sie die Lösung als kleinsten Fixpunkt.

Aufgabe 17

In einer Programmanalyse zur *Vorzeichenerkennung* werden alle negativen ganzen Zahlen durch das Symbol $-$ modelliert, Null durch 0 und alle positiven Werte durch $+$. So wird z.B. die Menge $\{-2, -1, 1\} \subseteq \mathbb{Z}$ durch $\{-, +\} \subseteq \{-, 0, +\}$ abstrahiert.

(a) Entwickeln Sie eine entsprechende Analyse für IC–Programme. Ordnen Sie hierzu jedem Block eines Programms $\pi \in IC$ eine (möglichst genaue) Information des Typs $V_\pi \to \mathfrak{P}(\{-, 0, +\})$ zu, welche die möglichen Vorzeichen jeder Programmvariablen angibt.

(b) Wenden Sie Ihr Analyseverfahren auf das folgende IC–Programm an:

$$
\begin{array}{rl}
 & \mathsf{out}\ z; \mathsf{loc}\ x, y; \\
1: & x \leftarrow 0; \\
2: & y \leftarrow -1; \\
3: & z \leftarrow x * y; \\
4: & \mathsf{if}\ x < y\ \mathsf{goto}\ 9; \\
5: & x \leftarrow x + 1; \\
6: & y \leftarrow y + x; \\
7: & z \leftarrow x * y; \\
8: & \mathsf{goto}\ 4
\end{array}
$$

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY

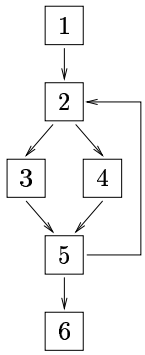
Sommersemester 2003
06.06.2003 (Abgabe: 20.06.2003)

Prof. Dr. K. Indermark
Dr. Th. Noll

7. Übung *Programmanalyse und Compileroptimierung*

Aufgabe 18

Gegeben sei das DFA–System $\Delta = \langle G; \mathfrak{F} \rangle$. Hierbei sei der Flussgraph G von der Gestalt



und die abstrakte Interpretation $\mathfrak{F} = \langle \langle \mathbb{B}^4; \leq^4 \rangle; \varphi; (0,0,0,0) \rangle$ sei gegeben durch die Transferfunktionen

$$\begin{aligned}
\varphi(1)(a,b,c,d) &= (0,b,1,d) \\
\varphi(2)(a,b,c,d) &= (a, b \vee c, c, d) \\
\varphi(3)(a,b,c,d) &= (a \vee d, b, b \vee c, d) \\
\varphi(4)(a,b,c,d) &= (a, 1, c, b \vee c \vee d) \\
\varphi(5)(a,b,c,d) &= (0,b,c,1) \\
\varphi(6)(a,b,c,d) &= (a,b,c,d)
\end{aligned}$$

Bestimmen Sie die MFP–Lösung von Δ, d.h. den kleinsten Fixpunkt der zugehörigen Gleichungstransformation.

Aufgabe 19

(a) Entwickeln Sie ein DFA–System zur Analyse der *Sprungstruktur* eines IC–Programms. Die mit dieser Analyse gewonnenen Informationen sollen für folgende Optimierungen genutzt werden können:

- Die Elimination unerreichbaren Codes, d.h. solcher Blöcke, die weder der Startknoten des Programms noch das Ziel einer Sprunganweisung sind.
- Die Entfernung von (bedingten oder unbedingten) Sprüngen zu unbedingten Sprunganweisungen.

(b) Analysieren Sie damit das folgende IC–Programm:

```
    in x; out y;
1 : if x > 0 goto 3;
2 : y ← 1;
3 : goto 6;
4 : y ← 2;
5 : goto 1;
6 : x ← x − 1;
7 : goto 5
```

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY

Sommersemester 2003
27.06.2003 (Abgabe: 04.07.2003)

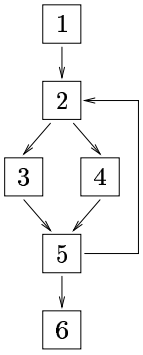
Prof. Dr. K. Indermark
Dr. Th. Noll

8. Übung *Programmanalyse und Compileroptimierung*

Aufgabe 20

Bestimmen Sie die MFP–Lösung des DFA–Systems aus Ü7A18 unter Verwendung des Worklist–Algorithmus aus der Vorlesung.

Zur Erinnerung: Der Flussgraph G ist von der Gestalt



Die abstrakte Interpretation $\mathfrak{I} = \langle\langle \mathbb{B}^4; \leq^4\rangle; \varphi; (0,0,0,0)\rangle$ ist gegeben durch die Transferfunktionen

$$\begin{aligned}
\varphi_1(a,b,c,d) &= (0,b,1,d)\\
\varphi_2(a,b,c,d) &= (a,b\vee c,c,d)\\
\varphi_3(a,b,c,d) &= (a\vee d,b,b\vee c,d)\\
\varphi_4(a,b,c,d) &= (a,1,c,b\vee c\vee d)\\
\varphi_5(a,b,c,d) &= (0,b,c,1)\\
\varphi_6(a,b,c,d) &= (a,b,c,d)
\end{aligned}$$

Aufgabe 21

Wenden Sie das in der Vorlesung diskutierte Verfahren zur *common subexpression elimination* auf das rechts stehende IC–Programm an.

```
    in x, y; out v, w;
1 : v ← x + y;
2 : w ← x * y;
3 : if w < x + y goto 7
4 : x ← x + 1;
5 : v ← x + y;
6 : goto 3;
```

Aufgabe 22

Ein *Bitvektor–System* ist eine spezielle Instanz eines DFA–Systems, bei dem

- der vollständige Verband gegeben ist durch

  $$\mathfrak{D} = \langle \mathfrak{P}(D); \leq\rangle,$$

  wobei D eine endliche Menge ist und $\leq$ entweder $\subseteq$ oder $\supseteq$, und

- bei dem zu jeder Transferfunktion $f : \mathfrak{P}(D) \to \mathfrak{P}(D)$ zwei Mengen $Y_f, Z_f \subseteq D$ existieren, so dass f sich wie folgt darstellen lässt:

  $$f(X) = (X \cap Y_f) \cup Z_f \qquad \text{für alle } X \subseteq D.$$

(Die Bezeichnung ist dadurch begründet, dass in diesem Fall jeder Analysewert durch einen Bitvektor repräsentiert werden kann.)

(a) Zeigen Sie, dass das DFA–System für *available expressions* (vgl. Vorlesung sowie A22) ein Bitvektor–System ist.

(b) Beweisen Sie, dass jedes Bitvektor–System distributiv ist.

(c) Geben Sie ein Beispiel für ein distributives DFA–System an, welches kein Bitvektor–System ist.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY

Sommersemester 2003
04.07.2003 (Abgabe: 11.07.2003)

Prof. Dr. K. Indermark
Dr. Th. Noll

9. Übung *Programmanalyse und Compileroptimierung*

Aufgabe 23

Eine Variante der *Available expressions*–Analyse ermittelt, ob ein Ausdruck *in einer bestimmten Variablen* zur Verfügung steht: Ein komplexer Ausdruck e ist in einem Knoten k des Datenflussgraphen verfügbar in einer Variablen x, falls e auf jedem zu k führenden Pfad an x zugewiesen wurde und falls danach weder x noch eine der Variablen in e modifiziert wurde.

Geben Sie ein DFA–System zur Berechnung dieser Analyseinformation an.

Aufgabe 24

Erweitern Sie die Analyse zur *Erkennung von Kopien* in SLC–Programmen (vgl. Ü2A5) zu einer Analyse für IC–Programme.

(Zur Erinnerung: Die Grundidee besteht darin, nach einer Kopieranweisung $x \leftarrow y$ nach Möglichkeit y statt x zu benutzen. Hierdurch wird die Kopieranweisung günstigstenfalls zu „Dead Code“ und kann anschließend eliminiert werden.)

(a) Konstruieren Sie ein entsprechendes DFA–System.

(b) Geben Sie eine darauf aufbauende optimierende Transformation für IC–Programme an.

(c) Demonstrieren Sie die Arbeitsweise Ihrer Analyse anhand eines geeigneten Beispiels.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY

Sommersemester 2003
11.07.2003 (Abgabe: 18.07.2003)

Prof. Dr. K. Indermark
Dr. Th. Noll

10. Übung *Programmanalyse und Compileroptimierung*

Aufgabe 25

(a) Geben Sie ein DFA–System an, welches zu einer gegebenen Schleife $S \subseteq Kno$ im Flussgraph eines IC–Programms die schleifeninvarianten Knoten bestimmt. Gehen Sie hierbei davon aus, dass der Graph in *single instruction*–Form vorliegt, d.h. dass jeder Knoten genau eine Anweisung enthält.

(b) Bestimmen Sie damit die invarianten Knoten in der Schleife des folgenden IC–Programms:

```
   in u; out y;
1 : v ← 0;
2 : if ... goto 10
3 : w ← u + 1;
4 : if ... goto 7
5 : x ← 2 * u;
6 : goto 8;
7 : v ← v + 1;
8 : y ← 3 * w;
9 : goto 2;
```

(c) Wenden Sie den den *code motion*–Algorithmus der Vorlesung auf das Programm in (b) an.

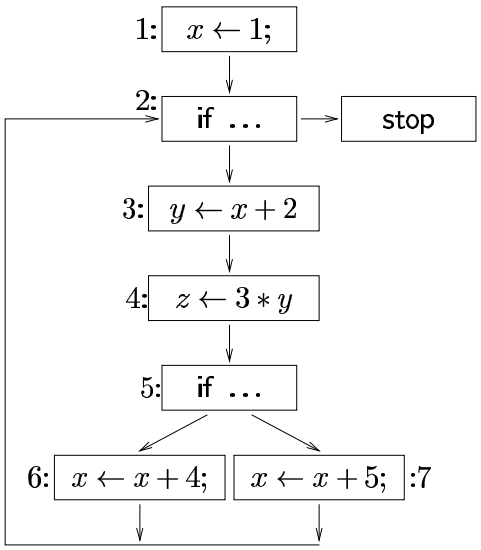
Aufgabe 26

In dieser Aufgabe soll der Eliminationsalgorithmus für Induktionsvariablen so erweitert werden, dass er auch abhängige Induktionsvariablen mit einer „additiven Definition“ (d.h. einer Zuweisung der Form $y \leftarrow x + c$ oder $y \leftarrow c + x$ mit $c \in \mathbb{Z}$, $x \in BIV(S)$ und $y \in AIV(S)$) erfasst. Gehen Sie hierbei wie folgt vor:

(a) Erweitern Sie das *strength reduction*–Verfahren auf additiv definierte, abhängige Induktionsvariablen.

(b) Zeigen Sie, wie eine Basisinduktionsvariable unter Verwendung einer additiven Induktionsvariablen eliminiert werden kann.

(c) Wenden Sie Ihren Algorithmus auf das rechts stehende Beispiel an.



RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY

Sommersemester 2003
18.07.2003 (Abgabe: 25.07.2003)

Prof. Dr. K. Indermark
Dr. Th. Noll

11. Übung *Programmanalyse und Compileroptimierung*

Aufgabe 27

Beweisen Sie, dass die folgenden drei Bedingungen aus der Vorlesung hinreichend für die Korrektheit der *Code Motion*–Optimierung sind. Diese gestattet das Verschieben einer S–invarianten Zuweisung $\alpha_l = l : x \leftarrow e$ vor den Anfang k der Schleife S, falls gilt:

(a) Für jeden Ausgangsknoten $k' \in S$ (d.h. es existiert $k'' \notin S$ mit $(k', k'') \in Kan$) gilt: l dom k'.

(b) Es gibt nur eine Definition von x ins S.

(c) Wird x in $l' \in S$ benutzt, so folgt $ud(x, l') = \{l\}$.

Weisen Sie hierzu nach, dass die Verschiebung von α_l vor k die Semantik des Gesamtprogramms nicht verändert.

Aufgabe 28

Entwickeln Sie ein Beispiel eines ICP–Programms, welches (anschaulich) belegt, dass die Ergebnisqualität einer Programmanalyse von der Modellierung des Kontrollflusses abhängt. Unterscheiden Sie hierbei die folgenden Fälle:

(a) Modellierung durch den „regulären“ Flussgraph (vgl. Folie 6.5)

(b) Modellierung durch den Standard–Flussgraph (vgl. Folie 6.4)

(c) Modellierung durch einen Flussgraph, der für jede (statische) Aufrufstelle einer Prozedur einen eigene Kopie des Flussgraphen der jeweiligen Prozedur bereitstellt

Hinweis: Es genügt, sich auf ein Hauptprogramm mit zwei Aufrufen einer nichtrekursiven Prozedur zu beschränken und als Anwendung z.B. die Konstantenpropagation zu betrachten.